DALLAS SEMICONDUCTOR MAXIM

Microcontroller
ENGINEERING REVIEW

第三卷

MAXQ™架构简介

MAXQ RISC架构结合了高性能和低功耗，以及多种复杂的模拟功能。

在为项目选择一款微控制器时，当今的微控制器系统设计者可以有数以万计的选择——8位、16位、RISC、CISC、或其他介于其间的微控制器。在选择过程中，需要考虑许多衡量标准，这已经成为了一种规则。这些衡量标准包括价格、性能、功耗、代码密度、开发时间、乃至未来升级的可能性等等。对一项标准的苛刻要求通常会影响到其他方面的选择，这就使选择更为复杂。某个因素在一种应用中至关重要，而在另一种应用中可能就不那么重要了。因此，没有一款微控制器可以做到完美地适用于所有项目。不过，一款现代的微控制器要想获得成功，就必须在许多目标领域里表现优越。

当世界著名的模拟芯片制造商 Maxim Integrated Products与业界领先的高性能微控制器供应商 Dallas Semiconductor联合时，便创造了一种机会，可以将高级模拟功能与前沿微控制器整合在一起。这种合作的结晶之一便是 MAXQ RISC架构，一种全新的微控制器核，结合了高性能和低功耗，以及多种复杂的模拟功能。

将复杂的模拟电路与高性能数字模块集成在一起时，工作环境必须尽可能安静和低噪。然而，微控制器核的数字电路中出现的时钟和开关会产生噪声，并注入敏感的模拟电路部分。这正是混合信号设计者所要面临的困难：既要赢得微控制器的高性能，又要将可能影响敏感的模拟电路的时钟噪声减至最小。

MAXQ架构通过智能化的时钟管理与利用降低了噪声。这意味着MAXQ核在任意时刻只向那些需要使用时钟的电路提供时钟，这样既降低了功耗，又为模拟电路的整合提供了一个最优的安静环境。另外，MAXQ架构在每个时钟周期都执行许多功能，使其性能得以最大化。本文纵览了MAXQ架构，并着重介绍其极具竞争性的优势。

不浪费任何时钟周期

MAXQ架构的设计目标之一是赢得更高的性能-功耗比。高效率设计的第一要素是最大限度地利用时钟周期来执行用户代码。

执行单周期指令是MAXQ获得高时钟利用率的最基本方法。执行单周期指令可以使最终用户受益，这是因为增加指令带宽可以获得更高的性能，并且/或者由于可以降低时钟频率，使得降低功耗成为可能。除了长跳转/长调用，以及某些扩展寄存器访问以外，所有MAXQ指令都在单个时钟周期中执行。尽管许多RISC微控制器都宣称支持单周期指令，不过通常只限于指令集的一小部分指令或寻址方式。而对MAXQ，单周期指令已成为规范。

其次，MAXQ架构实现了更高的时钟周期利用率，是因为它不借助指令流水线(常见于许多RISC微控制器中)实现单周期操作。MAXQ指令的解码与执行硬件极其简单(并且速度极快)，

目录

MAXQ架构简介 1
评测MAXQ指令系统和其它RISC竞争者 7
DS80C400的C语言编程.....17

...MAXQ核在任意时刻只向那些需要使用时钟的电路提供时钟，这样既降低了功耗，又为模拟电路的整合提供了一个最优的安静环境。

以至于这些操作可以与程序取指在同一时钟周期内完成，并且对最大工作频率的影响很小。为了解释省去指令流水线的好处，以一个普通的从流水线中执行指令的RISC CPU为例。当出现程序分支时，CPU需要花费一个或更多时钟周期(取决于流水线深度)来将取指操作转移到目标分支地址，并丢弃已取得的指令。显然，与直接执行指令相比，使用时钟周期来丢弃指令是不希望出现的浪费，因为这样降低了性能并增加了功耗。虽然用户不喜欢，CPU盗窃时钟去重装载流水线却是这种架构的必然后果，是无法避免的。MAXQ架构区别于其他8位、16位RISC微控制器的特点之一就是无须借助指令流水线实现单周期指令的执行(也就没有与之相伴的时钟周期浪费)。

MAXQ指令字

MAXQ指令字非常独特，因为只有一条传统意义上的指令，即“MOVE”指令。“MOVE”指令的源和目的操作数是生成指令、存储器访问以及触发硬件操作的基础。将16位MAXQ指令字分解后，得到的只有两部分：7位目的域，以及带有1位源格式位的8位源域。当源格式位编码为0时，表示以立即数或文本字节 (即 #00h–#FFh)作为源操作数。在单个指令字中对于立即数源的无限支持对于寄存器的初始化例程和ALU操作非常有价值。非文本源和目的操作数又细分为更小的组或模块。**图1**表示16位MAXQ指令字。

格式	目的	源
f	ddd dddd	ssss ssss
1	INDEX MODULE	INDEX MODULE
0	INDEX MODULE	IMMEDIATE BYTE DATA (i.e., 00h–FFh)

图1. MAXQ指令字很简单，但功能却很强大。

所有机器指令都归纳为用于传送操作的源和目的操作数。这些操作数被用来选择物理MAXQ设备寄存器。这种类型的传送是最基础且很容易想象的。不过，在MAXQ机器中，源和目的操作数并不一定与物理寄存器相关。

MAXQ架构使用这种相同的源到目的传送句法执行间接存储器访问。特定的目的和/或源编码被识别为物理存储器(例如堆栈、累加器阵列和数据存储器等)的间接访问入口。这些间接存储器访问入口使用物理指针寄存器来定义待访问的存储器地址。例如，间接访问数据存储器的方法之一就是使用“@DP[0]”操作数。以这种操作数分别作为源或目的，便可触发对于数据存储器的间接读或写访问，存储器由数据指针0 (DP[0])寄存器间接寻址。

MAXQ架构还使用专门的目的和/或源编码触发底层硬件操作。这种触发机制是生成那些与特定资源有隐性连接的MAXQ指令的基础。例如，算术运算(ADD、SUB、ADDC和SUBB)就是利用隐含目标为某个工作累加器的特殊目的操作数编码来实现的，用户只需提供源操作数。条件跳转的隐含目标是用于修改的指令指针(IP)，用于判断的各个状态条件具有独立的目的操作数编码。

存储器的间接访问和底层硬件操作的触发可在任何可能的情况下结合产生新的源/目的操作数。这就提供了两方面的优势。数据指针自动递增/递减的间接访问助记符正好代表了这种组合。当用DP[0]从数据存储器读取数据时，用户可以有选择地在读操作之后递增或递减指针，只需分别用“@DP[0]++”或“@DP[0]--”作源操作数即可。

MAXQ指令字的定义具有很多优点。指令字包含模块化分组的源和目的操作数，这样就简化并加速了指令译码硬件，并限制了与传送操作无关的模块中的信号切换，这样一来就降低了动态功耗和噪声。指令字采用全部16位来指定源和目的操作数，为物理寄存器、间接寻址存储器和硬件触发的操作提供了大量的地址空间。最终，大量的源/目的地址空间再加上对于源-目的组合的最少限制，造就了一个高度正交的设计。

...大量的源/目的地址空间再加上对于源-目的组合的最少限制，造就了一个高度正交的设计。

MAXQ系统纵览

MAXQ系统不仅提供当今微控制器用户所期望的基本硬件资源和能力，它还强化了这些资源，并增加了新的特性，以便扩展器件的功能和适用范围。在这里不可能列举所有的MAXQ系统资源，以下就讨论其中的一部分。

工作累加器

到现在为止，MAXQ架构作为一个单一实体我们已有所了解。不过，计划于2004年初投放市场的，MAXQ产品家族的最初成员，将是两个略有不同的版本，MAXQ10和MAXQ20。MAXQ10和MAXQ20的主要区别是工作累加器和算术逻辑单元(ALU)的标准宽度。MAXQ10支持8位(字节宽)累加器与ALU运算，而MAXQ20支持16位(字宽)累加器与ALU运算。MAXQ器件配备了最少8个累加器，而且，根据具体的应用，最多可以有16个累加器。在源/目的传送映像表中，这些累加器位于系统寄存器模块中，可以A[n]的形式单独访问每个累加器，其中*n*是其对应的标号。于是，配备了16个累加器的MAXQ器件包含累加器A[0]、A[1]、...、A[14]和A[15]。通过将累加器指针寄存器，AP，设置为某个特定标号，任何一个累加器都可以被指定为活动累加器，并通过*Acc*助记符间接访问(即Acc = A[AP])。AP寄存器的位数只要能够向累加器阵列提供足够的二进制译码即可，所以对于具有16个累加器的MAXQ器件而言四位就够了。所有ALU操作都隐含地指定活动的累加器作为正在执行操作的目的地。以“ADDC src”指令为例。该条指令实现活动累加器、进位标志和指定源操作数(src)之间的加法运算。还有很多位操作和移位/循环指令也是围绕活动累加器进行的。

累加器指针还配备了一些附属硬件，以便迅速、有序和可预见地访问累加器文件。利用累加器指针控制(APC)寄存器提供的控制位，我们可以对累加器指针寄存器AP进行复位、流线式递增、递减和取模运算。

处理器状态标志(PSF)寄存器包含5个状态标志，它们与活动累加器的状态和ALU操作相关。它们包括进位标志(C)、零标志(Z)、符号位(S)、相等标志(E)和溢出标志(OV)。这些标志中的很多可被作为执行条件跳转和返回的判断条件。PSF寄存器还为用户软件提供了两个附加的通用标志(GF1和GF0)。

为了实现高带宽、高效率和高正交性这一设计宗旨，MAXQ开拓性地采用了一种传送-触发架构。

专门的硬件堆栈

MAXQ架构包含一个专门的硬件堆栈。堆栈深度取决于具体MAXQ器件的型号。专门的硬件堆栈有两个突出的优点。首先，它使数据存储器保留给其他应用使用，而不会被堆栈消耗掉，其次，由于具有专门的读/写端口，不必和数据存储器分享总线，因而支持高速PUSH/POP操作。如果硬件堆栈的深度不足以满足保存现场之需，还可以利用数据指针的堆栈式工作方式(写时提前递增/递减，读时拖后递增/递减)在数据存储区中创建软堆栈。

MAXQ架构与其他8位、16位RISC微控制器的不同之处在于，它可以在不需要指令流水线(以及与之并存的被浪费的时钟周期)的前提下，提供单周期指令。

灵活的中断结构

MAXQ10和MAXQ20支持单一的、用户可配置的中断向量地址寄存器。这种方案允许用户按照自己的喜好布置中断识别和中断服务例程。任何中断源都没有强加的天然优先权。除了常规的个体和全局中断使能和标志外，还提供模块级的屏蔽和识别标志。单个中断源的使能、模块到全局级别的屏蔽，以及中断源的优先级等都可受用户代码的控制。这种中断支持结构有很多优势。首先，没有浪费代码空间。这通常是那些具有专门中断向量地址的微控制器所无法做到的，在这种微控制器中，未用中断向量所对应的代码空间通常被浪费掉了。其次，用户对于那些被使能的中断和中断的优先级有了更强的控制。

降低开销的硬件循环计数器

MAXQ架构的DJNZ指令可协同两个16位循环计数器(LC[0]或LC[1])中的任何一个工作。在单个时钟周期内，“DJNZ LC[n], src”指令就可递减循环计数器，并且，如果计数器未到0，就使程序运行分支到指定地址。对于其他RISC微控制器，循环计数器的更新与循环终止条件的检测通常是在两个单独的操作中实现。在MAXQ中这两个动作的合并意味着，软件循环这种微控制器应用代码中最常见的操作，MAXQ用更少的代码和时钟开销就可完成，因为不必再去管理循环计数器。单周期、DJNZ触发的循环计数递减与条件分支操作完全贯彻了我们最大限度利用时钟周期的宗旨。

强化的数据指针

MAXQ配备了3个16位数据指针(DP[0]、DP[1]和BP[Offs])。所有这三个数据指针都可以通过数据指针控制(DPC)寄存器中的字/字节选择位(WBSn)单独配置为字或字节访问模式。所有三个数据指针都支持单周期间接访问存储器，并且在写操作中具有提前递增/递减功能，在读操作中具有拖后递增/递减功能。其中一个数据指针被称为帧指针(FP = BP[Offs])，是由16位基指针(BP)寄存器和8位偏移(Offs)寄存器经过无符号加组合而成。这一类型的指针对于C编译器开发工具尤为重要，更具体地说是在堆帧的处理方面非常有用。

具有Von Neumann优势的Harvard存储器结构

MAXQ架构采用Harvard存储器组织方式，程序和数据存储器总线是分开的，这样在同一个时钟周期里可以同时访问指令字和数据字。要想赢得最高的性能、并在单个时钟周期内执行访问数据存储器的指令，这种存储器组织形式是必须的。采用Von Neumann存储器接口的微控制器会在总线上遭遇性能瓶颈，因为总线带宽要由程序存储器、数据存储器、I/O和外设共同分享。

Von Neumann架构的鼓吹者也会举出一些对方的弱点，例如不能像访问数据存储器那样访问程序存储器，反之亦然。具有这种访问能力可以简化常数存储、查表、在系统或在应用中进行编程等。MAXQ架构针对这个弱项的解决方案是加入一个存储器管理单元(MMU)和固定用途ROM，为用户提供逻辑存储器映射和支持在系统编程及满意的访问模式的固定用途例程。

对资源的集中访问

MAXQ架构的另一个重要特性是，出现了一个单一的传送映像表，它包含了到所有资源的接入点。之所以称其为传送映像表而非简单的寄存器映像表，是因为传送-触发概念正是建立MAXQ架构的基础。

传送映像表分成16个模块。每个模块中有32个索引号或独立的接入点。这里需要再次强调，这些接入点可用于对寄存器进行直接读/写访问，同时它们也可以用来间接访问存储器或触发硬件操作。在这16个模块中，前6个模块(M0–M5)被分配给器件特有的外设功能。这就为外设寄存器和访问入口在传送映像表中提供了宽绰的空间(6 x 32 = 192个位置)。基于特定的MAXQ器件选项，这些模块可以用来安排一些特殊功能寄存器，例如数字I/O、定时器、串口、硬件乘法器、LCD驱动器、ADC、以及在线调试器等。后10个模块(M6–M15)用于MAXQ系统功能。系统模块包含了对于MAXQ系统工作至关重要的寄存器，例如用于看门狗、系统时钟以及中断控制等的寄存器。系统模块还包含了工作累加器文件、数据指针，以及用来触发间接存储器访问和/或特殊操作的源/目的编码。在不同版本的MAXQ器件设计中，尽可能保持它们具有共同的基本系统寄存器空间。图2给出了MAXQ源和目的传送映像表的一个实例。

MAXQ包含一个单一的传送映像表，被分为16个模块，它提供到所有资源的接入点

前缀寄存器模块是MAXQ架构特别值得一提的特性。在MAXQ架构中存在着一个前缀寄存器，其中的数据(缺省值 = 00h)专供需要该数据的传送操作使用。装入数据后，该前缀寄存器只在一个时钟周期内保留该数据，然后就返回到00h态。前缀寄存器的选择必须伴随一个索引号(n)一起使用(PFX[n])。由于在传送映像表中共有16个模块，每个模块中又有32个索引号，因此，其中一部分就无法通过单个指令字提供的源/目的编码位直接访问(模块中的后16个源索引和后24个目的索引)。前缀寄存器打开了一个通向这些位置的窗口，从而解决了这个问题，这个窗口只保持一个时钟周期。当PFX[n]寄存器被加载时，其索引号“n”为紧随其后的指令提供高阶的源和目的编码位，*n* = *dds*。从这方面来讲，前缀寄存器模块提供了一种手段，它提供的附加译码位使我们能够访问扩展的(和/或受保护的)寄存器。装载前缀寄存器所需的操作和访问由汇编程序自动生成，不需要用户手动编制代码。当需要向16位目的地进行写操作时，前缀寄存器模块也可以用来和源字节相串联。前缀寄存器正是以这种方式被用于16位绝对地址的跳转和调用操作(这对于用户透明)。对于那些关心MAXQ架构未来发展的人来讲，前缀寄存器模块为MAXQ指令集的扩充、或向目前尚未使用的系统模块空间的延伸，提供了一条平滑前进的道路。

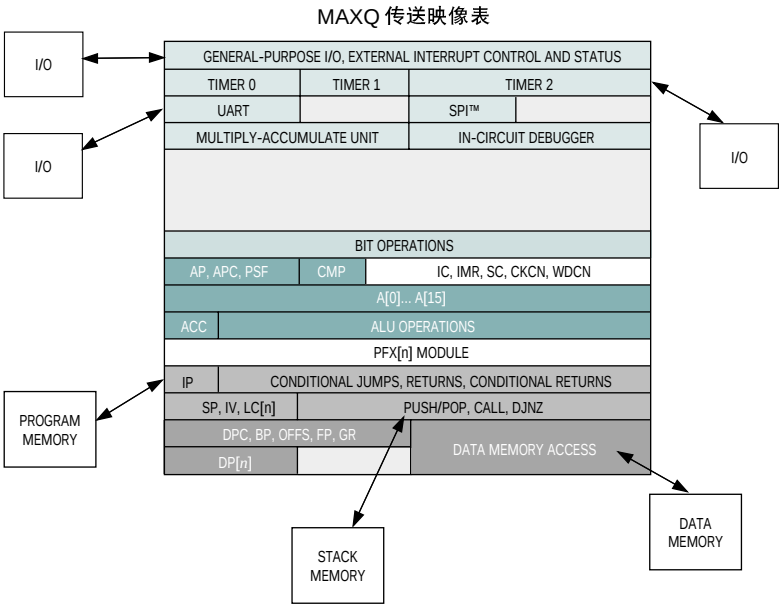

图2. 所有MAXQ资源可通过一个集中的传送映像表访问。

总的来讲，任何一款MAXQ器件的传送映像表都包含了为该器件所定义的全部系统和外设寄存器。同一个映像表可同时为数据存储区、堆栈存储区和累加器阵列提供间接接入点。同一映像表还包含了用于触发MAXQ机器指令和底层操作的接入点，并为未来的MAXQ家族指令集的简单扩展提供了条件。随着通向所有资源的接入点都汇集到了一个传送映像表中枢，源到目的传送的机会将是一个相当巨大的数量。对于资源集中化的访问也简化了按需分配时钟这一设计目标的实现，仅向那些需要时钟的资源提供时钟。这也提供了一个非常安静的环境(MAXQ中的“Q”–Quiet就是这个含义)，非常有利于模拟外设的集成。MAXQ架构最大限

...MAXQ系统和外设资源的模块化组织结构也导致了编译器的最优化，模块的可移植性加速了新的MAXQ衍生产品的诞生。

度实现了外设功能的模块化和可移植性。刻意采用这种策略是为了适应当今日益加速的产品开发周期，满足最终用户不断变化的外设需求，增加产品的灵活性和多用性。外设功能的模块化非常便于在新的设计中继承、添加或删除标准的MAXQ外设模块，当需要给某个特定市场或应用设计新的MAXQ器件时，可以节省大量的设计时间。

结论

MAXQ架构是当今微控制器业界的一个真正的创新。为了实现高带宽、高效率和高正交性这一设计宗旨，MAXQ开拓性地采用了一种传送-触发架构。此外，MAXQ系统和外设资源的模块化组织结构也导致了编译器的最优化，模块的可移植性加速了新的MAXQ衍生产品的诞生。放眼未来，MAXQ架构的内部特性为新一代产品的指令集扩展创造了条件。这些引人瞩目的优点使MAXQ架构成为当前和未来项目的理想方案，无论以何种标准来衡量，它都是一个名列前茅的选择。